NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN X11 GRAPHICAL INTERFACE FOR THE REPRESENTATION AND MAINTENANCE OF PROCESS KNOWLEDGE (REMAP) MODEL

by

Joseph A. Martinelli

September, 1993

Thesis Advisor: Balasubramaniam Ramesh

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Master's Thesis September 1991 to September 1993

4. TITLE AND SUBTITLE AN X11 GRAPHICAL INTERFACE FOR THE REPRESENTATION AND MAINTENANCE OF PROCESS KNOWLEDGE (REMAP) MODEL

5. FUNDING NUMBERS

6. AUTHOR(S) Joseph Anthony Martinelli

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13.
ABSTRACT *(maximum 200 words)*

The REpresentation and MAintenance of Process knowledge (REMAP) model provides support to various stakeholders involved in software projects by capturing the history of design decisions. This knowledge can assist the Department of Defense (DoD) in driving down the development and maintenance costs of large scale software systems. It is extremely important to have user friendly mechanisms to aid in the use of the REMAP model. This thesis implements a graphical user interface (GUI) under X11 Windows using the Andrew Toolkit. This implementation facilitates the instantiation, incremental modification, and ad-hoc querying of REMAP model primitives.

14. SUBJECT TERMS X Windows, Andrew Toolkit, REMAP, ConceptBase, GraphBrowser.

15. NUMBER OF PAGES 62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

An X11 Graphical Interface for the REpresentation and
MAintenance of Process Knowledge (REMAP) Model

by

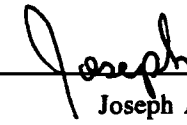
Joseph A. Martinelli
Lieutenant, United States Navy
B.S., Marquette University

Submitted in partial fulfillment
of the requirements for the degree of

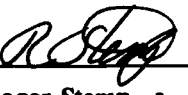
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

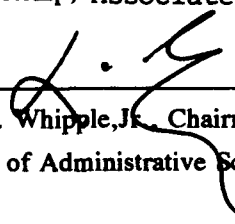
from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: 
Joseph A. Martinelli

Approved by: 
B. Ramesh, Thesis Advisor


Roger Stemp, Associate Advisor


David R. Whipple, Jr., Chairman
Department of Administrative Sciences

ABSTRACT

The REpresentation and MAintenance of Process knowledge (REMAP) model provides support to various stakeholders involved in software projects by capturing the history of design decisions. This knowledge can assist the Department of Defense (DoD) in driving down the development and maintenance costs of large scale software systems. It is extremely important to have user friendly mechanisms to aid in the use of the REMAP model. This thesis implements a graphical user interface (GUI) under X11 Windows using the Andrew Toolkit. This implementation facilitates the instantiation, incremental modification, and ad-hoc querying of REMAP model primitives.

TABLE OF CONTENTS

I. INTRODUCTION

A. GENERAL

The term graphical user interface (GUI) describes a user friendly interface that makes use of windows, menus, and other graphical objects and that, to a large extent, allows users to interact with the application by pointing and clicking mouse buttons (Barkakati, 1991, p. 92). In recent years, the demand for user friendly GUI software applications and bit-mapped graphics workstations necessary for their use have increased dramatically. These graphic intensive systems place a burden on developers due to the complexities of programming for bit-mapped screens (Brown, 1989, p. 1). The extensive use of the GUI interface has increased the complexity of programming tasks. Previously, each vendor had to develop a graphical interface for their particular hardware. The need to develop a different graphical interface for each hardware vendor was overcome by the X Window System (Jones, 1989, p. 1).

Another recent development is the rapid increase in multimedia systems. The essence of multimedia programs is that they permit the free intermingling of various ways of representing information (Borenstein, 1990, p. 2). This information can be full-motion, full-frame video with audio, fax, line drawings, text, etc. The Andrew Toolkit (ATK) is a

C-based toolkit which is used to aid the programmer in the development of multimedia programs. The Andrew Toolkit runs under the X Windows environment, allowing a wide variety of multimedia applications to be developed. The X Window System, or "X" as it is commonly called, provides a generic windowed graphics capability as a network service, accessible to any machine on the wire (Wolfe, 1992, p. 3). This allows for the elimination of hardware specific GUIs due to its portability across different platforms.

Support for various stakeholders involved in software projects can be provided by capturing the history about design decisions in the early stages of the systems development life cycle in a structured manner (Ramesh, 1992, p. 498). A conceptual model known as REMAP (**RE**presentation and **MA**intenance of **P**rocess knowledge) has been developed to allow the capture of knowledge gained from the design decisions made during the development process. This "captured" knowledge is known as process knowledge and its various benefits to the developers and maintainers of a system will be discussed in Chapter III.

This thesis is based on programming experience gained while developing a GUI for the REMAP model in the X Windows environment using the Andrew Toolkit.

B. THESIS OBJECTIVES

The primary objective in this research is to develop a graphical interface to the REMAP model. A user friendly GUI is essential for the successful use of a design rational management system. The REMAP model is implemented within *ConceptBase V3.2*, an experimental knowledge base management system developed at the University of Aachen. The implementation will extend the X-based application known as *GraphBrowser* that is provided within *ConceptBase* to incorporate REMAP specific functionalism. This implementation permits instantiation, modification, and querying of REMAP primitives using a graphical user interface.

There were two tasks necessary to achieve the primary objective of this thesis project. The first part was to modify an earlier C language implementation to allow customizable menu options for REMAP objects within the ATK-based *GraphBrowser* utility program. This would enable easy retrieval and manipulation of instances of REMAP objects. The second part was to incorporate functionalities for interactive queries, develop hypermedia links to REMAP objects, and to customize the *GraphBrowser* to display different REMAP objects with distinct characteristics.

C. APPLICABILITY TO THE DEPARTMENT OF DEFENSE

There is a need within DoD to drive down development and maintenance costs on all software development projects. The

REMAP model provides a vehicle that documents and maintains process knowledge which can be used to develop software more efficiently and economically as well as helping the maintainers of large systems understand their systems better, facilitating reduced maintenance costs over the life of a system. The incorporation of the REMAP model within the *GraphBrowser* utility function results in a program which is easily ported across different workstations, thus allowing its wide use in DoD software development projects.

D. SCOPE AND PREPARATION

The scope of this thesis is limited to a detailed review of the X Windows environment, the Andrew Toolkit, the REMAP model, and the *ConceptBase* program. A review of the previous work involving this project is also included. The remainder is a description of the design and implementation involved in extending the *GraphBrowser's* functionalities to incorporate the REMAP capabilities. Also addressed are the implementation of an ad hoc querying capability to access the selected knowledge base and the use of hypermedia to capture and maintain process knowledge.

Advanced knowledge of the C programming language and a basic understanding of X11 Windows and the Andrew Toolkit are required for this work. Preparatory work included an intensive course on C at the University of California at Santa Cruz, detailed review of literature on Andrew and extensive

e-mail exchange with the ConceptBase Development team at the University of Aachen, in Aachen, Germany.

E. ORGANIZATION OF THE STUDY

Beyond the introduction and a chapter on conclusions, this thesis consists of three major chapters. Chapter II provides an overview of the X Windows environment and the Andrew Toolkit. Chapter III will elaborate on the REMAP model and discuss some of its applications. A discussion of the *GraphBrowser* utility application and the *ConceptBase* knowledge base management system are also included as is a review of the background work accomplished on the REMAP implementation. Chapter IV will focus on the specific steps necessary to complete the REMAP model's integration into the *GraphBrowser* using the Andrew Toolkit.

II. OVERVIEW OF X WINDOWS AND THE ANDREW TOOLKIT

A. X WINDOWS IN GENERAL

In today's environment in which multiple vendors supply a wide variety of workstations, it is the consensus of the programming industry that there is a need to develop and field a windowing system which is compatible with a variety of machines. This thinking has become wide spread in the graphics workstation industry and has led to what is now known as the X Window System, a standard set of library routines which are implementable on all graphics workstations. The following is a description of how this system is designed and implemented as well as how several of the available toolkits are used to access the standard library routines.

1. Environment

The entire X environment is simply a series of layers built upon a base window system (see Figure 1). To interface with the remainder of the X Windowing system, the base window system must use the X network protocol. As shown, the network protocol is the only interface to the base window system (Jones, 1989, p. 2). Thus, software such as window managers and session managers are treated as application software which receive no special or privileged interfaces.

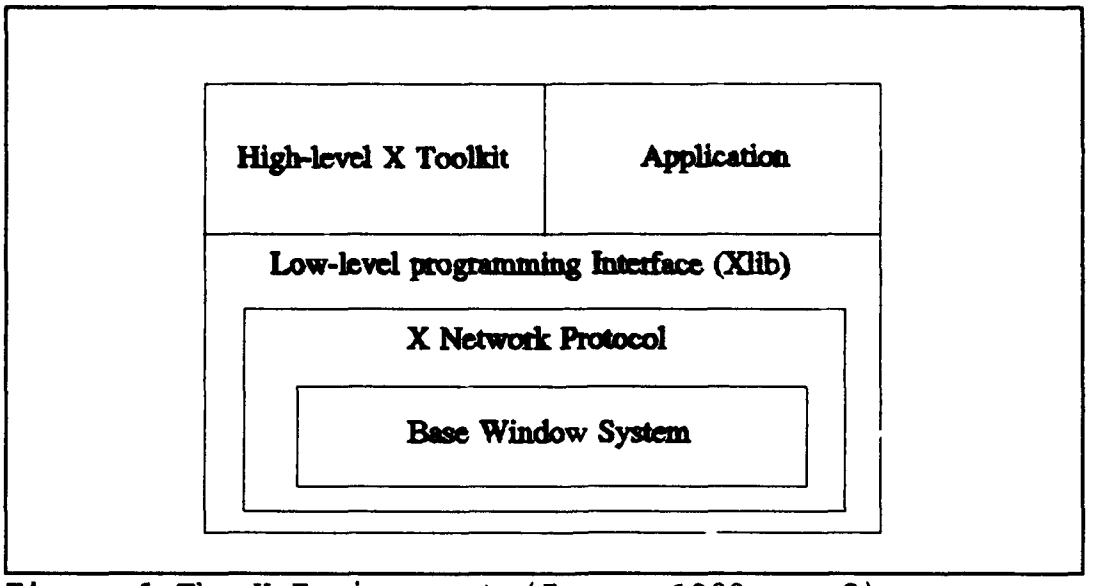


Figure 1 The X Environment (Jones, 1989, p. 2)

To access the network protocol, application programs use a C-language subroutine package known as Xlib. Xlib removes some of the complexities which could be involved when communicating with the network protocol and thus the base windowing system. To further mask some of the complexities of interfacing with the network protocol, high level X toolkits have been developed. These high level toolkits provide the routines to implement objects such as buttons, lists, and menus which can be used to build a graphical user interface (Barkakati, 1991, p. 12).

2. The Client-Server Architecture

The purpose of the X Window System is to provide a network-transparent and vendor-independent operating environment for workstation software (Jones, 1989, p. 4). The network transparency mentioned above refers to the ability of

an application to run on whatever central processing unit (cpu) is convenient and display that output on the desired machine. This transparency is achieved through the use of a well defined protocol, called the X network protocol, (Barkakati, 1991, p. 5).

The X network protocol uses a programming model known as the client-server model (Barkakati, 1991, p. 5). The process executing on your local workstation is called the server and manages the graphics output and the input from the keyboard and mouse, while the client is the process which utilizes the capabilities of the server workstation (Barkakati, 1991, p. 7). It is important to note that the definitions of "client" and "server" are just the opposite of those used in LAN environments and mini-computer environments (see Figure 2).

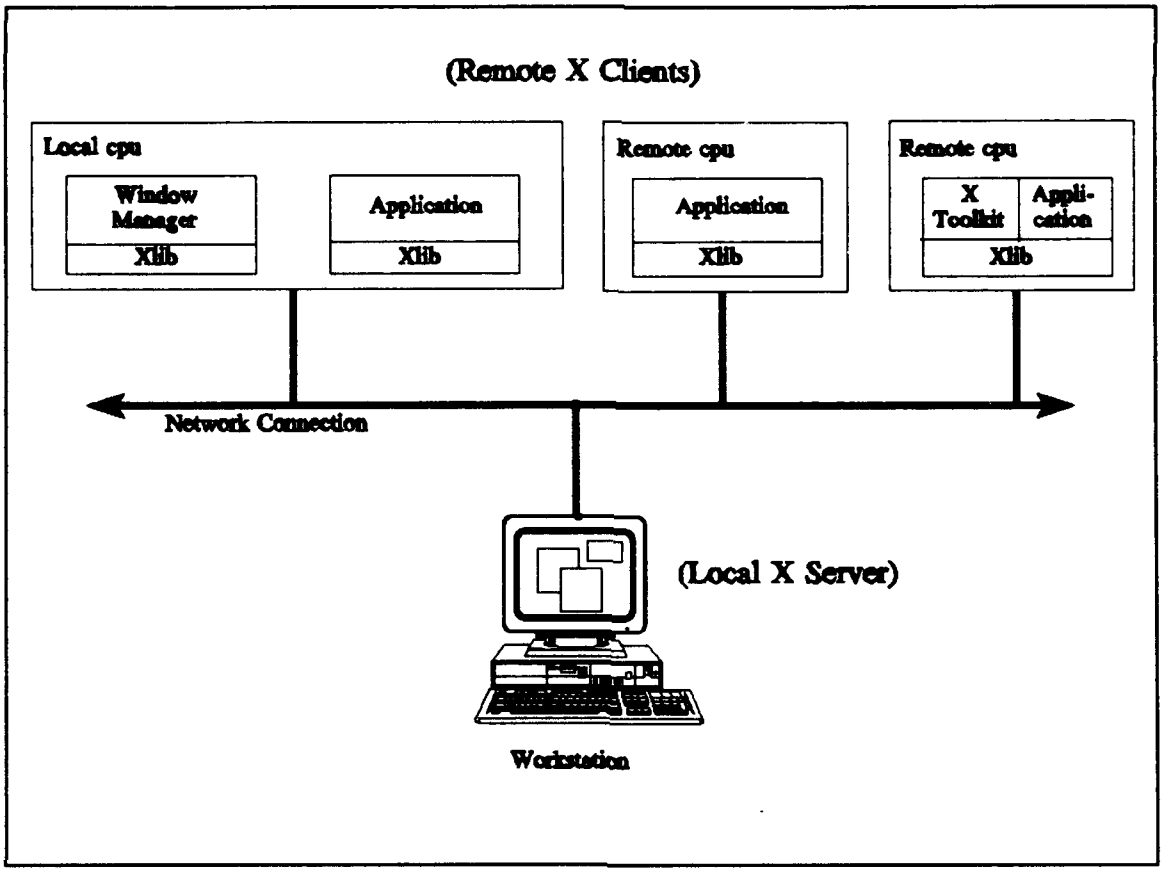


Figure 2 X Client-Server Architecture (Jones, 1989, p. 4)

B. THE ANDREW TOOLKIT

The Andrew System is a multi-function system developed at Carnegie Mellon University. It is composed of three main components:

1. The Andrew Message System.
2. The Andrew Help System.
3. The Andrew Toolkit and Application Programs.

This section will focus on the Andrew Toolkit and some of the attributes which make it useful to our work.

The Andrew Toolkit is a system written to assist programmers in designing and building multimedia programs and was implemented using the C programming language. Although the C language itself does not support the object oriented structure which is desirable for multimedia programming, the Andrew Toolkit includes an inheritance mechanism to allow true object oriented programming. Also included within the Toolkit are a preprocessor to allow high-level abstractions to be compiled and a dynamic linking mechanism.

1. Object Orientation

To achieve object oriented programming, the Andrew Toolkit has established object classes. Essentially, an object class is an abstractly defined generic object. Thus, objects used in the programming of the Andrew Toolkit are actually specific instances of an object derived from a larger class. An Andrew Class is composed of two files: the standard C file (".*c*") containing the class data and methods, and a class header file (".*ch*") containing the class specification.

a. Compilation of an Andrew Application

The class header file is loaded into the ATK preprocessor, a program known as "class." The class program takes ".*ch*" files as input and produces two files as output:

- The ".*eh*" (exported header) file is included by the class implementation.
- The ".*ih*" (imported header) file is included by any other code that wants to use a class. (Borenstein, 1990, p. 20)

Since ".*c*" files are not loaded into the preprocessor, there are no intermediate versions of the source code to be dealt with prior to compilation. This allows changes to be easily made to the source code besides simplifying the debugging.

The ".*c*", ".*eh*", and required ".*ih*" files are loaded in to a standard C language compiler (cc) which generates an object file (".*o*"). The object files are loaded into a program called "makedo" which outputs a dynamically loadable ".*do*" (dynamic object) file. The resultant ".*do*" can then be loaded by an application at run time.

2. Dynamic Loading

Dynamic loading is the ability to load a part of your program into memory after the main program has been initiated. This allows each class which the program is using to be independently loaded as needed. This results in several important benefits:

1. It allows programs to be linked in substantially shorter periods.

2. It permits easier code sharing.

3. It makes the ATK as a whole more easily extended or modified because it does not require recompiling of the basic ATK code.

4. It results in a great gain in run-time efficiency on most versions of the UNIX operating system. (Borenstein, 1990, p. 21)

However, care must be exercised by the user when shared header files change. All objects which depend on those header files must be recompiled.

a. The runapp Program

The primary program to manage dynamic loading in the Andrew Toolkit is the runapp program. It is a single binary program which holds all of the toolkit objects together. For an application to be run within the ATK, the runapp binary program must first be started and then the desired specific applications dynamically loaded. If runapp is to be run under an application known as "remap", a symbolic link is formed between remap and runapp. A subclass of remap called "remapapp" is located within the dynamic object file *remap.do* and is then dynamically loaded. Figure 3 illustrates the complete compilation and dynamic loading process.

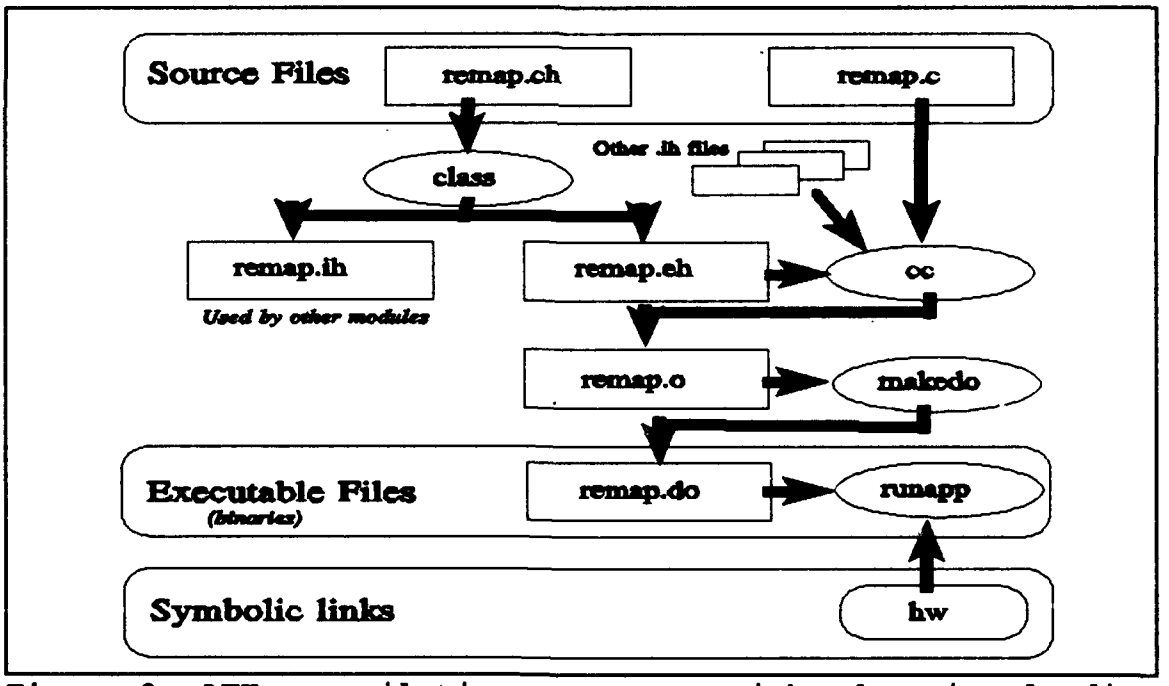


Figure 3 ATK compilation process with dynamic loading (Borenstein, 1990, p. 25)

3. Insets

In order to facilitate the sharing of an object among a wide range of applications, the Andrew Toolkit implements a something called the inset. An inset is actually a pair of ATK objects conforming to a specified set of conventions designed to make them easily used by almost any Andrew Toolkit application (Borenstein, 1990, p. 56). The two objects which make up an inset are the data object, which is a subclass of the basic class called *dataobject*, and the viewing object, which is a subclass of the basic class called *view*. Basically, the data object contains the data which makes up the object to be displayed. The viewing object contains the information necessary to actually display the object as well as the specific ways a user can interact with and manipulate the object.

Insets provide a number of advantages. The use of insets allows multiple views to be attached to a single object. For instance, one view could look at a series of numbers as a table while the other view could look at the series of numbers as a pie chart. By associating multiple views with a common object, a programmer can modify one visible object and any other visible object associated with the common object will also be modified. This flexibility enables the programmer to use an object in a number of different programs just by altering the *dataobject/view* relationship. The use of insets provides a very modularized

programming environment which promotes code reusability. This also encourages application development because elaborate and complex application objects are replaced by smaller, simpler objects which can be more easily built and reused (Borenstein, 1990, p.57).

For a more complete description of the characteristics of X Windows and of the Andrew Toolkit, refer to Nabajyoti Barkakati (Barkakati, 1991), Oliver Jones (Jones, 1989), and Nathaniel Borenstein (Borenstein, 1990).

III. REMAP ENVIRONMENT

A. REMAP BACKGROUND

The focus of the REMAP project is the capturing of history of system development in the upstream part of the life cycle (Ramesh, 1992, p.498). The history which REMAP captures concerns the design decisions/rationales which shape the design of the software development project. This design rationale is typically lost in the course of designing and changing a system (Ramesh, 1992, p. 498). Some uses of the process knowledge captured by the REMAP model and used by system designers, system maintainers, and system users are:

- facilitate communication of information sharing among various stakeholders in a project.
- reduce system maintenance efforts by maintaining knowledge at the requirements and design rationale level.
- allow end users to understand how exactly the design-deliberation process addresses their requirements. (Ramesh, 1992, 508)

1. The REMAP Component

The REMAP conceptual model incorporates the Issue Based Information System (IBIS). The IBIS method was developed by Horst Rittel and is based on the principle that the design process for complex problems is fundamentally a conversation among the stakeholders in which they bring their respective expertise and viewpoints to the resolution of

design issues (Conklin, 1988, p. 304). IBIS was used at the Microelectronic Computer technology Corporation (MCC) in the Design Journal research project as a way of representing design deliberations in large design projects (Ramesh, 1992, p. 499). The IBIS method utilizes a set of three primitives and relationships among them in a rhetorical model for representing the conversation process.

The three primitives comprising the IBIS model are labelled Issue, Position, and Argument (see the region within the dashed line in Figure 4). An Issue is a question or concern which requires discussion before the problem solving can proceed. A Position is a statement or assertion which responds to an Issue. An Argument is a statement that supports or objects to a Position. An Argument can have more than one Position associated with it. (Ramesh, 1992, p. 500)

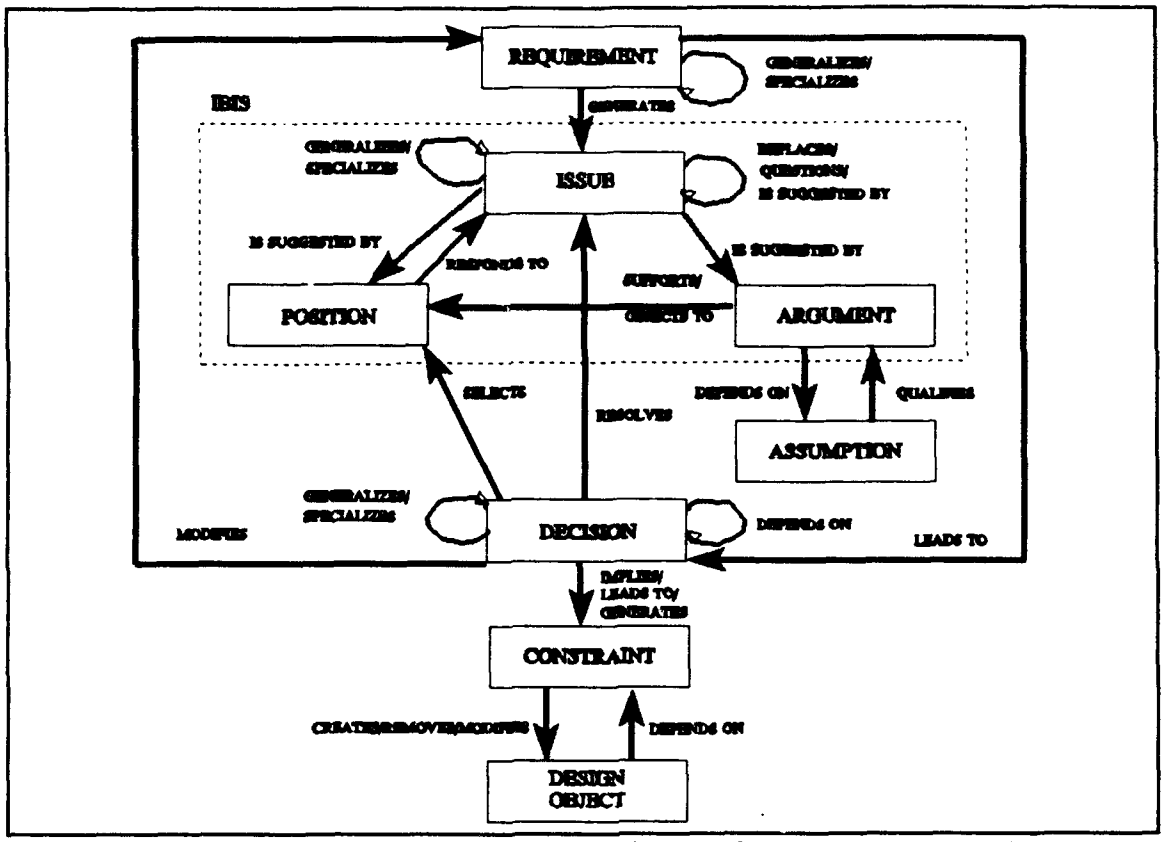


Figure 4 REMAP Conceptual Model (Ramesh, 1992, p.501).

2. What Makes Up REMAP?

The initial IBIS model was meant to capture the conversations/discussions between stakeholders in a project, but it did not recognize the context in which "argumentations" occurred and it did not take into account the results of these "argumentations" (Ramesh, 1992, p. 500). The REMAP model incorporated five additional primitives: Requirement, Decision, Assumption, Constraint, and Design Object, whose relationships can also be seen in Figure 4. When a system is being designed, the end goal is to satisfy the user's requirements. Thus, Requirement is the embodiment of what the

users desire. The REMAP model shows that Requirements are subject to change after deliberation. REMAP incorporates Assumptions to allow for mechanisms to evaluate the applicability of an Argument in a given situation. A Decision represents the resolution of one or more Issues through which designers establish constraints or commitments that must be accounted for in the final design. The final design is labelled Design Object.

B. REMAP IMPLEMENTATION WITHIN *CONCEPTBASE*

The REMAP model is implemented within *ConceptBase*, a deductive object base management system (Jarke, 1993, p. 1). *ConceptBase* uses the knowledge representation language Telos which is an object-oriented conceptual modelling language. *ConceptBase V3.2* is implemented in a client-server architecture where the clients and servers run as independent processes which interact via inter-process communication (IPC). The *GraphBrowser* program provided within *ConceptBase* is written entirely in the C programming language using the Andrew Toolkit. This utility is intended to provide a customizable GUI. *ConceptBase* also incorporates several utilities such as a Telos Editor and a Hypertext Editor.

1. *GraphBrowser* Utility Function

The *GraphBrowser* is a windows oriented utility function which allows a selected knowledge base to be viewed graphically. Any displayed object can be selected by clicking

on the object with the left mouse button. Each object possesses an object identifier which the *GraphBrowser* uses to draw the initial node of a directed acyclic graph (DAG).

- **erase node** - deletes the selected object (node or link) from the display.
- **any** - displays all objects connected to the selected object by a link corresponding to a user-defined specification of orientation and label.
- **show attributes** - displays all the direct attributes.
- **show instances** - displays all the direct instances.
- **show classes** - displays all classes the selected object is an instance of.
- **show subclasses** - displays one level of subclass.
- **show superclasses** - displays one level of superclass.
- **Help** - displays a window containing a description of the *GraphBrowser's* facilities.
- **quit** - terminates the GraphBrowser. (Jarke, 1993, pp. 20-21)

Various menu choices trigger *GraphBrowser* queries which retrieve objects from the knowledge base according to four pre-defined Telos graphical types: *MetaMetaclass*, *MetaClass*, *SimpleClass*, and *Token*.

C. PRIOR WORK

Earlier efforts to implement REMAP used the Andrew Toolkit version 5.0 and *ConceptBase V3.0*. Although the implementation of the REMAP model was incomplete, the Andrew Toolkit interface to the *ConceptBase* system was coded and debugged.

The final product allowed the *ConceptBase* program to be called, the *GraphBrowser* utility function to be initiated, and some fundamental calls made to the desired knowledge base using REMAP structured calls.

1. Noted Deficiencies and Possible Solutions

In the effort, the *GraphBrowser* application was not customized to provide a distinctive graphical scheme for REMAP objects. Also, the complete REMAP model was not incorporated into the knowledge base.

The availability of a more advanced version of *ConceptBase* (version 3.2) provided functionalities to overcome these deficiencies. Capabilities essential for an implementation of the REMAP model such as a customizable graphical display for the *GraphBrowser* and hypermedia capabilities are included. Additionally, a new query mechanism (*CB_ASK_GBFORMAT*) that allows the user to specify how links and nodes will be displayed in the *GraphBrowser's* window within a given *ConceptBase* session is also provided.

IV. IMPLEMENTATION DETAILS

A. STANDARD *GRAPHBROWSER* INTERFACE

The *GraphBrowser* application is written as a layer around a set of shared objects. The Andrew Toolkit assists in using shared objects through the creation of insets. As explained in Chapter II, an inset is a pair of objects, a data object and a viewing object, that displays the data object on the screen (Borenstein, 1990, p. 57). The *GraphBrowser* inset consists of the data object *cbGraph* and the viewing object *cbGraphView*.

The class *cbGraph* stores the part of the semantic network to be displayed and provides methods for manipulating it . These methods include inserting and deleting objects, accessing the fields stored in the instances of a class, setting some of these fields, and computing the neighbors of an object. The methods *cbGraph_Insert*, *cbGraph_Delete*, *cbGraph_SetCB_ID*, *cbGraph_SetName*, and *cbGraph_Neighbours* , respectively, perform these functions. (Eherer, 1992, p. 13)

Once an object has been defined, it must be determined how to view it. This is done by *cbGraphView*. The class *cbGraphView* maintains the data used for displaying the objects as well as methods which draw the object or scroll the window. The methods *cbGraphView_WantUpdate*, and

cbGraphView_WantInputFocus perform these functions. (Eherer, 1992, p. 14)

The basic *GraphBrowser* allows a knowledge base to be graphically accessed. However, the basic *GraphBrowser* does not incorporate the object definitions of the REMAP model or customized menus to provide REMAP specific functionalities. Achieving these goals requires separate programming tasks. The first task consists of the coding of a C based interface using the Andrew Toolkit to allow the direct manipulation of a REMAP knowledge base. The second task centers around extending the *GraphBrowser* utility function to incorporate the REMAP model and call the created interface.

B. CREATING THE NEW ANDREW TOOLKIT EXTENSION

The creation of an extension using the Andrew Toolkit is a two step task. First, a new Andrew class file must be written. Then, this file must be compiled so that it can be made available to the *GraphBrowser* utility function. Only minor changes to the existing code were necessary to make it compatible with the newer version of the *ConceptBase* system. This section provides only a brief overview of this task. For additional description of the major steps, the reader is referenced to the thesis by J.J. Stenzoski (Stenzoski, 1992).

1. Writing the Andrew Class File

The Andrew class file is composed of two portions, the class header file (".*ch*") and the standard ".*c*" file. A brief overview of these follows.

a. Class Header File

The class header file is similar, but not identical, to normal C include files (".*h*") (Borenstein, 1990, p. 16). A new class is defined by giving it a name and establishing how it will differ from its superclass. Our class will be called *remap*. The *remap* class does not need to inherit any procedures, so our new class is a top-level class (Stenzoski, 1992, p. 24). Here is the *remap* class header file, *remap.ch*:

```
#define remap_VERSION 1

class remap {
    classprocedures:
        InitializeClass () returns boolean;
        InitializeObject (struct remap *self) returns boolean;
        FinalizeObject();
};
```

This class header file contains the procedure specification for the initialization of the *remap* class and object instance as well as for cleaning up and freeing memory when the object is deleted (Stenzoski, 1992, p. 24).

b. The Class C File

The corresponding ".*c*" file contains the actual procedures which were specified in the ".*ch*" file (refer to Appendix B for the ".*c*" code).

The first section of the ".*c*" file is the *#include* files. This section includes the various files needed by the routine to perform the desired functions as well as the functions needed to provide input/output and interaction capabilities. The *remap.eh* file is the export file created from the *remap.ch* file by the Class preprocessor. It enables the other Andrew classes to utilize the procedures of the *remap* class.

The next section contains the C code written to extend the *GraphBrowser*. The two procedures *remap_query* and *remap_instanceof* are created and defined here. These are the procedures that will be called by the extended *GraphBrowser*. *remap_query* shows the particular dependencies for an object entered in a interaction window. *remap_instanceof* displays the representation of an object entered in an interaction window.

The *InitializeObject* and *FinalizeObject* are class methods required for proper object instantiation and termination (Stenzoski, 1992; p. 26).

The *InitializeClass* method makes the call to the *proctable_DefineProc* procedure. This enters the internal procedure names *remap-query* and *remap-instanceof* into the *cbGraphView's* procedure table. The procedure table is used to establish and translate bindings between menu items and their corresponding procedures.

2. Compiling the New Andrew Class

An *Imakefile* which uses macros developed especially for the Andrew system was used. The reader is referred to book by Nathaniel Borenstein (Borenstein, 1990) for an explanation. The *Imakefile* is a *Makefile* generator. The *Makefile* is used to compile the program resulting in executable code. The overall compilation process required the setting of environment variables, the creation of the *Imakefile*, and the code compilation.

a. Setting Environment Variables and Creating the Imakefile

The environment variables establish the paths which will be necessary to complete the compilation and to establish the required dependencies. This includes the location of the dynamic objects required to complete compilation. These variables are set in the *.cshrc* file of the project directory with the values shown below:

```
setenv ANDREWDIR /usr/local/andrew51
setenv PATH .:$ANDREWDIR/bin:$PATH
setenv CLASSPATH .:$ANDREWDIR
```

The following *Imakefile* was used for the compilation process:

```
NormalObjectRule()
DependTarget()
NormalATKRule()
DynamicObject(remap,,)
InstallClassFiles(remap.do, remap.ih)
CC = gcc
PICFLAG = -fpic
```

b. Makefile Generation and Code Compilation

The first step is generation of the *Makefile*. This is done by issuing the following command from the project directory:

% genmake

Once the *Makefile* is generated, the file dependencies must be established. This is accomplished with the following command:

% makedepend

To complete the compilation and create the "*.do*" file type:

% make

The Andrew Toolkit extension is now ready to be run by the extended *GraphBrowser*. Once the Makefile is generated and the dependencies established, only the final command of the above process has to be used if a change is made to the "*.c*" file.

C. EXTENDING THE *GRAPHBROWSER* UTILITY FUNCTION

There are two steps involved in the extension to the GraphBrowser utility function. The first step involves the modification of the *.graphbrowserinit* file to allow the extension (detailed above) to be called. The second step involves the coding and loading of the models necessary to implement the REMAP model. These models will allow for the correct identification and graphical representation of the selected objects.

1. Modifying the *.graphbrowserinit* file

The *.graphbrowserinit* file is composed of two sections: the menu item description section and the graphical type description section. The first section describes the buttons available within the *GraphBrowser's* panel. The second defines the mapping from graphical types defined within the knowledge base to concrete patterns and shapes displayed on the screen (Eherer, 1992, p. 21). The basic *.graphbrowserinit* file contained within the *ConceptBase V3.2* release is shown below:

```
#include <gb_init.h>
Erase~60,GB_MULTISELECT,gb-gb_Erase
Add Node~61,GB_UNSELECT,gb-gb_AddNode
Any~62,GB_SOMETHING,gb-gb_any
Show Attributes~63,GB_SOMETHING,gb-gb_show_attributes
Show Instances~64,GB_NODE,gb-gb_show_instance
Show Classes~65,GB_NODE,gb-gb_show_classes
Show Subclasses~66,GB_NODE,gb-gb_show_subclasses
Show Superclasses~67,GB_NODE,gb-gb_show_supclasses
Telos Editor~70,GB_MULTISELECT,gb-gb_CallEditor
Graph Browser~71,GB_MULTISELECT,gb-gb_CallGraphBrowser
Save Layout~81,GB_NOTHING,gb-SaveLayout
Load Layout~82,GB_NOTHING,gb-LoadLayout

dummy,none,oval,yellow,GB_NODE|GB_SOMETHING|GB_TOOLS
```

Descriptions of the menu item and graphical item descriptions are below. The modifications necessary for each section are provided.

a. Menu Item Description

Menu items are included at the beginning of the *.graphbrowserinit* file and consist of: menu item, menu mask, and menu procedure (Eherer, 1991, p. 8). The first entry is the

string which will appear as a menu selection item to the right hand side of the *GraphBrowser* window when the *GraphBrowser* is called. The number following the tilde (~) gives the relative position of the menu item on the screen. The higher the number is, the lower that menu item will appear on the screen. To place a space between groupings of menu items, the number must jump up to the next decade (e.g., 60's up to 70's).

The second entry in the line is the menu mask. Menu masks may be given as defined in the file *gb_init.h* or as numbers directly in the init file (Eherer, 1991, p. 8). The *gb_init.h* file distributed with *ConceptBase V3.2* is shown below:

```
#define GB_NOTHING          0L
#define GB_NODE             1L
#define GB_LINK             2L
#define GB_SOMETHING        4L
#define GB_UNSELECT         8L
#define GB_TOOLS            16L
#define GB_MULTISELECT      32L
```

A boolean evaluation is performed between the menu item's mask and the selected object's mask to determine if a menu item will be displayed (Jarke, 1993, p. 22). The object mask is determined in the graphical item portion of the *.graphbrowserinit* as described below. If the result of the boolean evaluation is true, the menu item is displayed.

The final entry is the name of the procedure which the menu item calls. The routine *remap_query* , mentioned earlier, is an example of a procedure which may called to execute. However, to execute the call correctly, the Andrew system requires the routine to be called as *remap-query*.

To include the two REMAP procedures mentioned in the *GraphBrowser* with the names "Input Remap Object" and "Show Remap Instances," the *.graphbrowserinit* file must look like this:

```
#include <gb_init.h>
Erase~60,GB_MULTISELECT,gb-gb_Erase
Add Node~61,GB_UNSELECT,gb-gb_AddNode
Any~62,GB_SOMETHING,gb-gb_any
Show Attributes~63,GB_SOMETHING,gb-gb_show_attributes
Show Instances~64,GB_NODE,gb-gb_show_instance
Show Classes~65,GB_NODE,gb-gb_show_classes
Show Subclasses~66,GB_NODE,gb-gb_show_subclasses
Show Superclasses~67,GB_NODE,gb-gb_show_supclasses
Telos Editor~70,GB_MULTISELECT,gb-gb_CallEditor
Graph Browser~71,GB_MULTISELECT,gb-gb_CallGraphBrowser

/*          APPENDED MENU ITEM                  */
Input Remap Object~72,GB_NODE,remap-query
Show Remap Instances~73,GB_NODE,remap-instanceof

Save Layout~81,GB_NOTHING,gb-SaveLayout
Load Layout~82,GB_NOTHING,gb-LoadLayout

dummy,none,oval,yellow,GB_NODE|GB_SOMETHING|GB_TOOLS
```

For either of these two menu items to appear, the item selected must have an object mask equal to GB_NODE.

b. Graphical Item Description

The second section of the *.graphbrowserinit* file determines how a retrieved object is displayed and what menu items will be available when that object is displayed. A line in this section of the *.graphbrowserinit* file consists of five parts: GraphicalType, Pattern, Shape, Color, MenuMask. The GraphicalType is the name of the Telos class which the GraphBrowser uses to identify the retrieved object. Pattern is the fill/connection style (none, black, gray, dashed_link, double_link) used when the retrieved object is displayed. Shape

identifies how the retrieved object will be formed (rectangle, oval, circle, diamond, icon, link) when it is displayed. Color determines the outline color for the object. Lastly, MenuMask determines which menu items will be available when the object is displayed and selected. These masks are separated by a "|" which represents a bit-wise OR. This bit-wise OR allows flexibility in determining which menu items are available for each graphical type. (Jarke, 1993, p. 22)

To accommodate the REMAP model, graphical type descriptions for REQUIREMENT, ISSUE, POSITION, ARGUMENT, ASSUMPTION, DECISION, CONSTRAINT, AND DESIGN_OBJECT are required. The resulting *.graphbrowserinit* file is shown below:

```
#include <gb_init.h>
Erase~60,GB_MULTISELECT,gb-gb_Erase
Add Node~61,GB_UNSELECT,gb-gb_AddNode
Any~62,GB_SOMETHING,gb-gb_any
Show Attributes~63,GB_SOMETHING,gb-gb_show_attributes
Show Instances~64,GB_NODE,gb-gb_show_instance
Show Classes~65,GB_NODE,gb-gb_show_classes
Show Subclasses~66,GB_NODE,gb-gb_show_subclasses
Show Superclasses~67,GB_NODE,gb-gb_show_supclasses
Telos Editor~70,GB_MULTISELECT,gb-gb_CallEditor
Graph Browser~71,GB_MULTISELECT,gb-gb_CallGraphBrowser
Input Remap Object~72,GB_MULTISELECT,remap-query
Show Remap Instances~73,GB_MULTISELECT.remap-instanceof
Save Layout~81,GB_NOTHING,gb-SaveLayout
Load Layout~82,GB_NOTHING,gb-LoadLayout

/*              APPENDED REMAP GRAPHICAL TYPES                */
REQUIREMENT,none,circle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
ISSUE,none,oval,black,GB_LINK|GB_SOMETHING|GB_TOOLS
POSITION,none,rectangle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
ARGUMENT,black,rectangle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
ASSUMPTION,none, rectangle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
DECISION,gray,rectangle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
CONSTRAINT,black,oval,black,GB_LINK|GB_SOMETHING|GB_TOOLS
DESIGN_OBJECT,black,rectangle,black,GB_LINK|GB_SOMETHING|GB_TOOLS
```

Once an object is displayed within the *GraphBrowser* window, it can be designated by clicking on it using the left mouse button. This will allow any of the various commands available at the right side of the GraphBrowser window to be applied to the desired object.

ConceptBase V3.2 allows further customization of how the graphical types can be displayed. Various attributes of a graphical type (such as background color, shape and text color) can be specified by creating a Telos model of these specifications. The file *colors.sml* (Figure 5) specifies the color an object is shaded, the color of the text within the object, and the shape of the object for various REMAP objects.

```
REQUIREMENT with
     attribute
          ColorWithShape : "bubblenode";
          ColorBackgroundColor : "blue";
          ColorTextColor : "white"
end

ISSUE with
     attribute
          ColorWithShape : "ovalnode";
          ColorBackgroundColor : "pink";
          ColorTextColor : "black"
end

POSITION with
     attribute
          ColorWithShape : "rectnode";
          ColorBackgroundColor : "green";
          ColorTextColor : "black"
end

ARGUMENT with
     attribute
          ColorWithShape : "diamondnode";
          ColorBackgroundColor : "yellow";
          ColorTextColor : "black"
end

ASSUMPTION with
     attribute
          ColorWithShape : "rectnode";
          ColorBackgroundColor : "orange";
          ColorTextColor : "black"
end

DECISION with
     attribute
          ColorWithShape : "rectnode";
          ColorBackgroundColor : "red";
          ColorTextColor : "black"
end

CONSTRAINT with
     attribute
          ColorWithShape : "ovalnode";
          ColorBackgroundColor : "purple";
          ColorTextColor : "white"
end

DESIGN_OBJECT with
     attribute
          ColorWithShape : "rectnode";
          ColorBackgroundColor : "brown";
          ColorTextColor : "white"
end
```

Figure 5 *colors.sml* Model

Any of the colors available to the X Windows system can be used to customize the output. Other attributes which can be interpreted by the *GraphBrowser* include LineWidth (width of the lines of nodes and links), LineColor (color in which the outline of the selected objects is drawn), and LineDash (dash factor between 0(solid line) and 16(dashed)). The prefix Color is placed in front of each attribute for use with color monitors and the prefix mono is used for use with monochrome displays.

2. Creating The New REMAP Model

The definition of new graphical types that have been defined within the *colors.sml* file must be made available to the *ConceptBase* server. This is done by creating two new classes, *RemapEDGE* and *RemapNODE*, which can be found in the file *RemapEGDE.sml*.

To create the new REMAP specific classes, the original *GraphBrowser* classes *X11GraphBrowserEDGE* and *X11GraphBrowserNODE* were used as building blocks. The original *X11GraphBrowserEDGE.sml* file used the MetaMetaClass, MetaClass, SimpleClass, and Token graphical types. The REMAP extension requires the addition of the eight graphical types used in the *.graphbrowserinit* REMAP definitions into the *RemapEDGE* and *RemapNODE* classes.

The *GraphBrowser RemapEDGE* class specification after the addition of the eight REMAP graphical types is below:

```
Class RemapEDGE in AnswerRepresentation isA EDGE with
    graphicalType
        gT1  : REQUIREMENT;
        gT2  : ISSUE;
        gT3  : POSITION;
        gT4  : ARGUMENT;
        gT5  : ASSUMPTION;
        gT6  : DECISION;
        gT7  : CONSTRAINT;
        gT8  : DESIGN_OBJECT
end
```

The next step in creating the new REMAP specific classes involved ordering the graphical types. Ordering the graphical types establishes a level of precedence. Because the Telos modelling language allows objects to be assigned to more than one class, an object called by the *GraphBrowser* can be of more than one type. The *GraphBrowser* selects the graphical type with the lowest number (hence the highest precedence) and uses that graphical type to display the object. The *orderValue* definitions of the *RemapEDGE* class are shown below in Figure 6:

```
RemapEDGE!gT1 with
orderValue
        v:1
end

RemapEDGE!gT2 with
orderValue
        v:2
end

RemapEDGE!gT3 with
orderValue
        v:3
end

RemapEDGE!gT4 with
orderValue
        v:4
end

RemapEDGE!gT5 with
orderValue
        v:5
end

RemapEDGE!gT6 with
orderValue
        v:6
end

RemapEDGE!gT7 with
orderValue
        v:7
end

RemapEDGE!gT8 with
orderValue
        v:8
end
```

Figure 6 *RemapEDGE.sml* Model

The same ordering performed for the *RemapEDGE* class must also be performed for the *RemapNODE* class.

The new classes must now be made available to the *GraphBrowser*. This requires the *setenv* command to set *CB_ASK_GBFORMAT* to *ON*. By setting the *CB_ASK_GBFORMAT* variable to *ON*, an interaction window is called which prompts the user for the node and edge formats which the *GraphBrowser* will use. In this implementation, the node and edge formats required to bring in the REMAP model are *RemapNODE* and *RemapEDGE*, respectively.

3. Loading the RemapEDGE and colors Models

Now that the two new files *RemapEDGE.sml* and *colors.sml* have been created, they must be added to the *ConceptBase* knowledge base. When the new models are loaded and the Andrew Toolkit extension is compiled, the extended *GraphBrowser* is ready for use. Appendix A provides a short tutorial on how to get started on a session with the extended *GraphBrowser*.

D. HYPERMEDIA AND AD-HOC QUERY EXTENSIONS

The *Graphrowser* utility function is only one of the many capabilities of *ConceptBase V3.2*. A hypertext capability which allows the incorporation of text and audio clips is also included. Furthermore, an ad-hoc querying capability is built in which allows selective viewing of the knowledge base. The following will give a more in depth description of how these functions can be used to extend the REMAP model's functionality.

1. Hypertext Editor

The Hypertext editor allows the editing of hypermedia documents. Hypermedia documents can include text (such as Andrew Toolkit "*.ez*" documents), audio clips, raster graphics, vector graphics (poor), tables, and video clips. The current version of the Andrew Toolkit can work with all of the above mentioned hypermedia documents with the exception of video clips.

Hypermedia capabilities provide many possibilities for the REMAP project. The gathering of data to document and preserve the process knowledge can become obtrusive in a project and this may discourage the use of the model. The need to use a graphical tool during a deliberation session can cause disruptions in the work flow, thus hampering group work. One way to overcome this is through video taping and audible recording of the "argumentations" which occur during group meetings. A "scribe" may be assigned the task of indexing these multimedia sources in terms of the REMAP model.

a. Sample Session with the Hypertext Editor

The model presented below (Figure 7) represents the portion of the REMAP model in which Issue is defined to have a definition which is a hypermedia object. Two instances of Issue are presented: priority_scheme and frequency. To each of the instances, a hypermedia definition is attached. These definitions are provided in files created with the Andrew

Toolkit hypermedia editor, EZ. In this example, the files are text files which provide a definition of what the particular instance is. This could be easily extended to include audio clips.

```
ISSUE with
 attribute
  definition: HyperMediaObject
end

priority_scheme with
 attribute
  definition: priority_scheme_definition
end

Individual priority_scheme_definition in
HyperMediaObject with
 filename
  : "/project1/priority_scheme.ez"
end

frequency with
 attribute
  definition: frequency_definition
end

Individual frequency_definition in
HyperMediaObject with
 filename
  : "/project1/frequency.ez"
end
```

Figure 7 Hypertext model

A new user of the REMAP model is examining a project for the first time. The user notices something referred to as a priority_scheme but does not understand what it is. The hypertext editor can be used to extract this information. By selecting "Editing" from the Tool Bar, then choosing "Hypertext Editor" from the available items, an interaction window appears. The string

"priority_scheme_definition" is entered. A new window (Figure 8) appears with the definition of a priority_scheme.

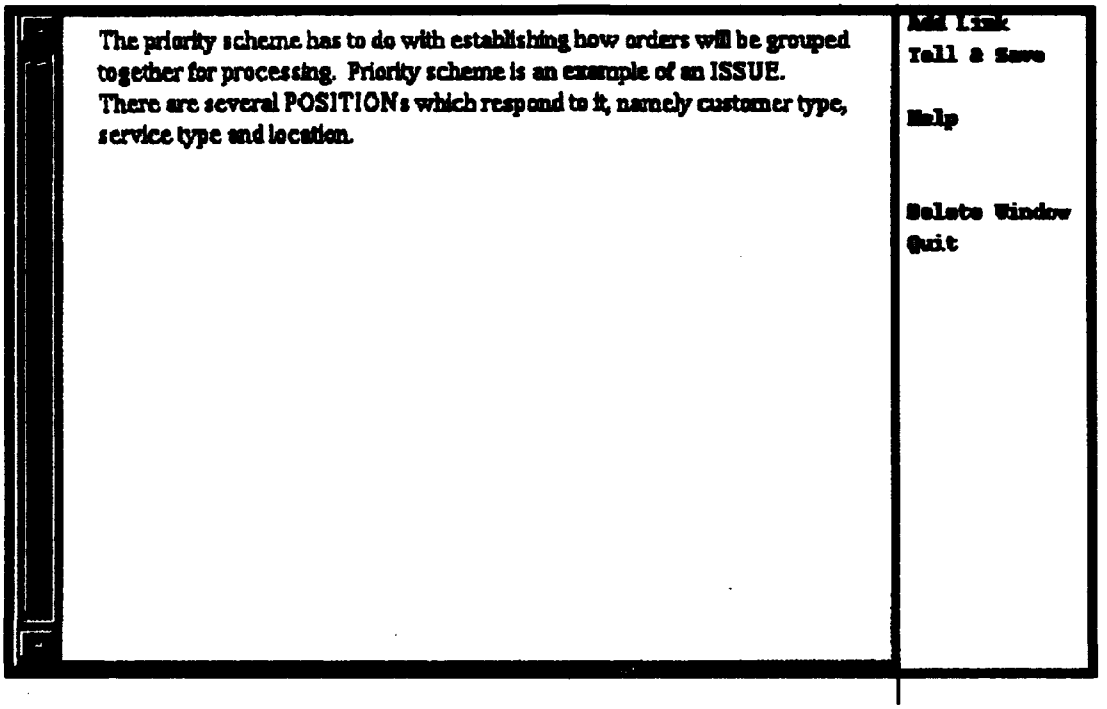


Figure 8 Hypertext Answer

2. Ad-Hoc Querying

The ad-hoc querying capability of *ConceptBase* can be a very useful tool within the REMAP environment. As a software development project grows, the size of the knowledge base required to maintain the process knowledge for all the various stakeholders can become quite large. The information needs of various participants may vary widely. For instance, the needs of a manager will differ greatly from those of a designer or maintainer. Rather than having to browse the entire knowledge base to retrieve relevant information, an ad-hoc query can be structured using the GUI to retrieve the necessary information. The information will be presented in a structured manner.

The model below (Figure 9) represents Telos queries to retrieve a portion of the knowledge base which a manager would be interested in. QueryClasses are formed to support the different stakeholders needs. In this example, the QueryClasses OpenIssues and ResolvedIssues are established. These queries will assist a manager tracking the progress of various Issues concerning the different aspects of a project. The first query will retrieve all Issues whose status attribute is set to PENDING and the second query retrieves all RESOLVED Issues. It should be noted that *ConceptBase* provides deductive database capabilities through which the status of an

Issue may be dynamically inferred based on the status of other model components.

```
QueryClass OpenIssues isA ISSUE with
  constraint
    pending: $ this status PENDING $
end

QueryClass ResolvedIssues isA ISSUE with
  constraint
    pending: $ this status RESOLVED $
end
```

Figure 9 Display Queries Model

a. Sample Session Using Ad-Hoc Queries

A program manager is interested in what issues have been resolved. To obtain this information, the ad-hoc query generator is called. This is accomplished by selecting "Displaying" from the Tool Bar, then selecting "Display Queries" from the available choices. An interaction window (Figure 10) appears displaying the various QueryClasses available. The manager selects ResolvedIssues by highlighting the item, then chooses "Ask" to send the query. A new window (Figure 11) appears showing that the only issue to have been resolved up until this point is priority_scheme.

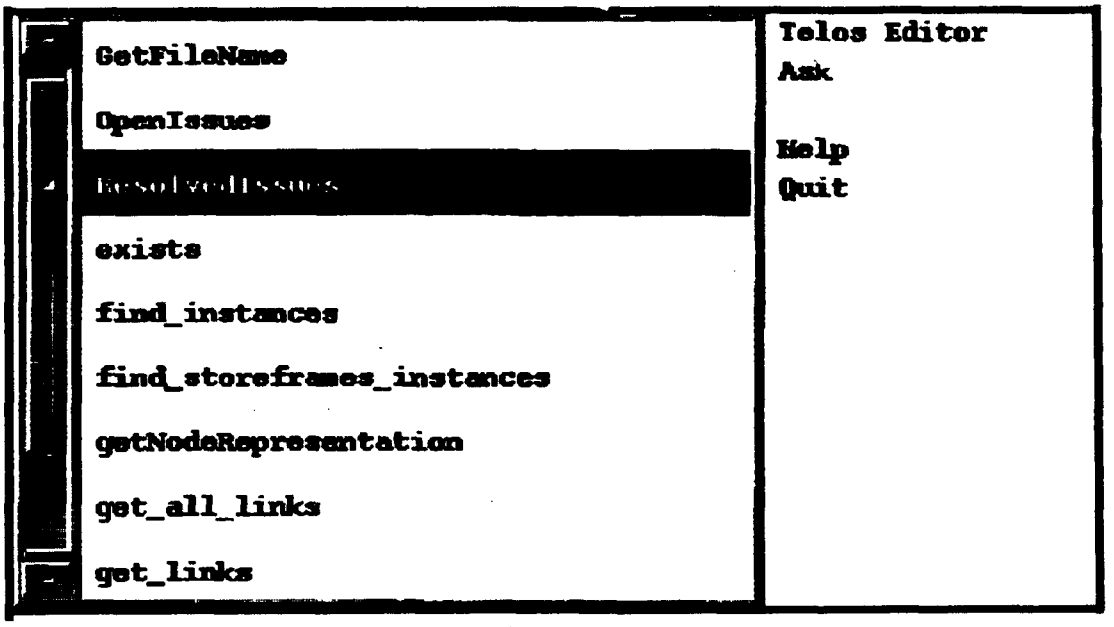


Figure 10 QueryClass Specializations

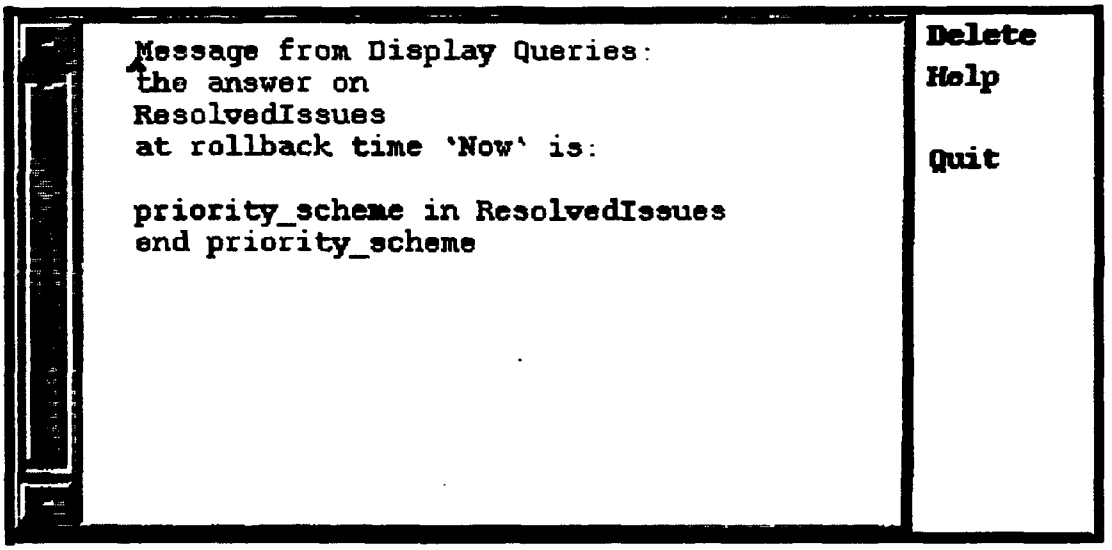


Figure 11 Display Queries Answer

V. CONCLUSIONS AND RECOMMENDATIONS

A. PROGRAMMER REQUIREMENTS

The learning curve for a programmer using the Andrew Toolkit for the first time is quite steep. There are a number of skills necessary for performing any type of development with the Andrew Toolkit. First, a good understanding of the Unix operating environment is required. Both *ConceptBase* and the Andrew Toolkit make extensive use of *Makefiles* to manage system development. A good understanding of the C programming language is also necessary prior to attempting to program with the Andrew Toolkit. Pointers and arrays are used extensively. The basics of the X Window system are helpful as far as understanding the client-server architecture and how it can be used. The most valuable skill was experience gained by actually working with the system. Invaluable knowledge was gained over hours of experimenting with the *ConceptBase* program and its various components (including the *GraphBrowser*).

B. HELPFUL REFERENCES

The author found that the only comprehensive book about the Andrew Toolkit was Nathaniel Borenstein's *Multimedia Applications Development with the Andrew Toolkit*. This book was written with a more experienced programmer in mind and a

good background in the C programming language is a prerequisite for understanding. Additional help on the Andrew Toolkit can be found as on-line documentation within the Toolkit. Further questions can be directed to the Andrew Toolkit Consortium at Carnegie Mellon University that can be contacted at:

Andrew Toolkit Consortium
Carnegie Mellon University
4910 Forbes Avenue
Pittsburgh, PA. 15213-3890
(412) 268-6700 / FAX (412) 621-8081

Mailing list: info-andrew@andrew.cmu.edu
Newsgroup: comp.soft-sys.andrew

It was the author's experience that the best source of information and guidance could be obtained from the *ConceptBase* developers at the University of Aachen, in Germany. A substantial portion of this implementation was made possible by the e-mail clarification and suggestions provided by the ConceptBase design team at the University of Aachen. They can be contacted at:

ConceptBase-Team
c/o Rene Soiron
RWTH Aachen - Informatik V
D-52056 Aachen - Germany

+49-241-80-21-501 / FAX +49-241-80-21-529
e-mail : CB@picasso.informatik.rwth-aachen.de

C. RECOMMENDATIONS

A programmer fluid in the C programming language, familiar with the working of the Andrew Toolkit, and possessing a good

understanding of Telos is required to continue this project. Without the need to overcome the steep learning curve for this environment, such a person could complete additional capabilities into the REMAP model implementation within *ConceptBase* in a matter of months.

The ConceptBase program is still in development. No capability exists to easily delete models which have been added to the Knowledge base. Also, there is insufficient documentation to assist a programmer engaged in an effort such as ours. Such work requires an extensive working e-mail relationship with the University of Aachen.

D. CONCLUDING COMMENTS

ConceptBase V3.2 is a very powerful program. Although the full functionality of the REMAP environment was not developed, the necessary building blocks have been provided.

The REMAP model, when fully implemented and functional in the *ConceptBase* environment, will prove a great benefit to the Department of Defense. Capture of process knowledge with the REMAP model will benefit development and maintenance of systems. Its uses range from assisting in the development of new systems to the re-engineering of systems already in place.

APPENDIX A : GRAPHBROWSER TUTORIAL

The following was extracted from the Appendix of J.J. Stenzoski's thesis (Stenzoski, 1992). It has been modified to incorporate the additional functionality provided by *ConceptBase V3.2*. Screen snaps taken during an actual session are shown to help the first time user.

Running the *GraphBrowser* from a Remote Workstation

1. Login and run X Windows by entering <**start**> or <**xinit**>.
2. In the console window, enter <**xhost +**>. The advisory "all hosts being allowed (access control disabled)" will appear.
3. In an X terminal, **rlogin** to the workstation where the *GraphBrowser* and extension is installed. This terminal will hereafter be referred to as the GB terminal.
4. In the GB terminal, enter <**setenv DISPLAY** <local machine address>**:0.0**> to send *GraphBrowser* graphics to the local workstation.
5. In another X terminal, **rlogin** to the machine running *ConceptBase* (this terminal will be referred to as the CB terminal).
6. Continue with "Procedures Common for Local *GraphBrowser* Operation."

Procedures Common for Local *GraphBrowser* Operation

1. Start the *ConceptBase Toolbar* in the GB terminal by entering **<toolbar>**. The Toolbar (Figure 12) window should appear within a few seconds. **<toolbar>** is an alias for the command "CBinterface" from the directory $CB_HOME/CB_Exe/XI_Exe/sun4.

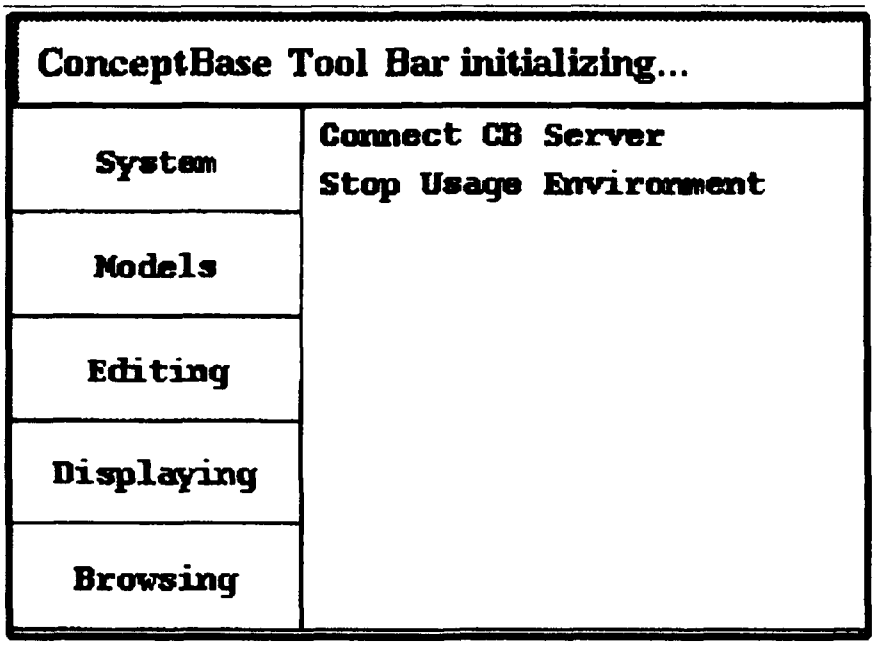


Figure 12 The Basic Tool Bar

2. If a second **xterm** window has not already been started, start one now to use as the CB terminal. In the CB terminal, start the *ConceptBase* server by entering **<cbserver>**. **<cbserver>** is an alias which calls up the command "CBserver -d ~/project/New App" from the directory $CB_HOME/CB_Exe/KSV_Exe. This starts the *ConceptBase* server with the database NewApp. When the server is started, note the portnumber that the server is "ready under" (typically 4001).

3. In the Toolbar window, display the SYSTEM menu by first clicking on the SYSTEM box with the left mouse button to designate the SYSTEM tool. Then, select "connect CB server" by highlighting the "connect CB server" menu item and once again clicking the left mouse button. An interaction window will appear asking for the host machine and the portnumber (Figure 13). Ensure that the portnumber assigned matches that assigned in step 2. If it does not match, correct this discrepancy prior to continuing.

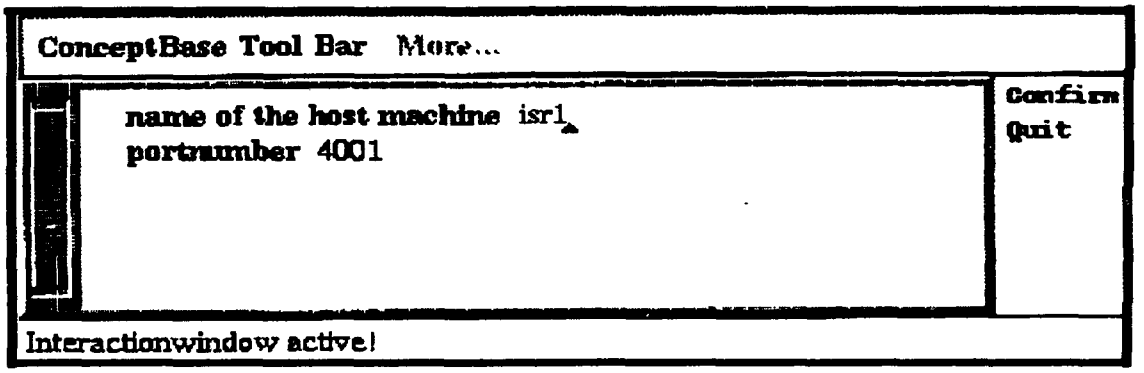


Figure 13 Portnumber Interaction Window

4. From the BROWSING menu card, select *Browse graphically* using the technique described in step 3 above. An *Interaction Window* will appear and request the name of the object to browse. Enter **<process_order>**. You will then be queried for the edge and node definitions to be used. Enter **<RemapEDGE>** for edge type and **<RemapNODE>** for node type. See Figure 14 below. The comment "The GraphBrowser starts up. Wait a little bit for its window." will appear in the *Toolbar* comment area.

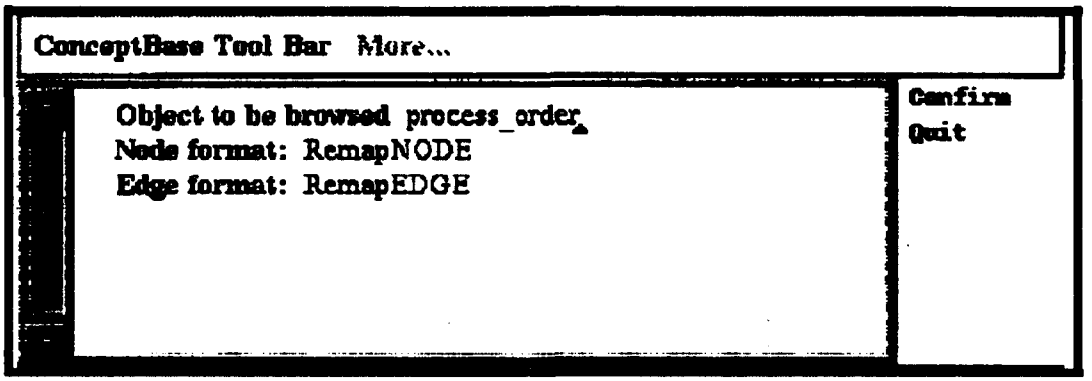


Figure 14 Node and Edge Format Interaction Window

5. The *GraphBrowser* window will appear with the process_data object graphically displayed. Operations on displayed objects can be performed by selecting the object with the left mouse button, then selecting the desired operation from the menu item list to the right of *GraphBrowser* window.

6. To stop the *GraphBrowser*, from the SYSTEMS menu card select "stop CB server." From the same SYSTEMS menu card, select "stop usage environment."

APPENDIX B : REMAP.C CODE

```
/*  Extended GraphBrowser application for the REMAP project.

Author:  LCDR J. J. Stenzoski, US Navy

Follow-on Authors : LT Joseph A. Martinelli, Stefan Eherer

Initial revision:  16 September 1992
Follow-on revision : 20 August 1993                          */


/************************** Include files ******************************/
#include "remap.eh"        /* External header file */
#include "cbgraphv.ih"     /* CB GraphBrowser view file */
#include "cbGraph.ih"      /* CB GraphBrowser data object file */
#include "proctbl.ih"      /* Procedure Table file */
#include "iw_main.h"       /* interactionwindow file */
#include "iw_interact.h"   /* interactionwindow file */
#include "iw_conchecks.h" /* interactionwindow error checker file */
#include <stdio.h>
#include "CB_interf. .h"

/***************************Function definition**************************/
/* Declares gb_parser function as "extern" */
extern int gb_parser(struct cbGraphView *self, char *returndata, int type);
Server *server;
char* portnumber;
int port;

/* declaration of "query" function */
void
remap_query(struct cbGraphView *self, long rock)
{
/************************** Variable declaration ************************/
char *source;              /* name of source */
char *label;               /* name of link or node */
char *dist;               /* distance from source node (currently limited to 1)*/
char long_string[300];    /* string variable to capture get_links query */
char *rd;                 /* server's response to "ask" query */
char **input;             /* user's interactionwindow entry */
int typ = 0;              /* argument for gb_parser call (specifies how objects
                             are to be drawn on the screen) */

/********** Generate interaction window and obtain user input **********/

/* defines "entry" data structure with interactionwindow prompt in quotes */
struct interaction entry  = {"Enter object to retrieve:", NULL,
                                                  NULL, NULL};

/* declares an array of "entry" structures with pointer "entries" */
struct interaction *entries[] = {&entry, NULL};

/* changed the data structure of the current object to graphitem SE */
struct graphitem *gi;

printf ("entering ...");
```

```
gi = cbGraphView_GetCurrentObject(self);
printf ("done!\n");

/* assigns user input to variable "input" and checks for NULL input */
if (!(input=interactionwindow(entries)) || input[0]==NULL)
   return;

/************************ Query server for user-input object*****************/

/* sprintf copies the entire get_links query into the variable "long_string.
   In the get_links query, self->currentObject->CB_ID designated the
currently selected object and is substituted for the first %s argument,
while *input is
substituted for the second %s argument in the query.  *input will contain the
user's input from the interactionwindow. */

/* test code prior to the call to the knowledge base*/
/* used the ID of the graphitem instead of the ID of the current object SE */
printf("Variable values prior to the call : rock= %ld, name = %s\n",
rock, graphitem_RetID(gi));

/* used the ID of the graphitem instead of the ID of the current object SE */
sprintf(long_string,
"get_links([coming_out_of/dir,'%s'/source,%s/label,1/dist]),FORMAT:RemapE
DGE",
                                graphitem_RetID(gi), *input);

/* queries server with get_links above and assigns response to "rd" */
rd = ask(server, long_string);

/* test code after the call to knowledge base*/
/* used the ID of the graphitem instead of the ID of the current object SE */
printf("Variable values after the call: rock= %ld, name = %s\n", rock,
graphitem_RetID(gi));


/* checks for errors in server response */
if(strcmp(rd, "nil") == 0 || *rd==ERRORCHAR)
  {
       return;
  }
/******** Parse server response and display object instance in GB window *****/
/* sends server response to the parser routine for graphocal display */
gb_parser(self, rd, typ);

} /* end query procedure */

/************* remap_instanceof Procedure *****************/

void
remap_instanceof(struct cbGraphView *self)

{
char *rd;
int typ = 0;
char longer_string[300];

/* changed the data structure of the current object to graphitem SE */
struct graphitem *gi = cbGraphView_GetCurrentObject(self);

/* instead of plugging using user input in for %s as the "label" as in the
remap_query procedure, here "instanceof" will be used as label whenever the
procedure's menu item is selected. No interactionwindow is used. */
/* used the ID of the graphitem instead of the ID of the current object SE */
sprintf(longer_string,
"get_links([coming_out_of/dir,'%s'/source,'*instanceof'/label,1/dist]),FO
```

```
RMAT:RemapEDGE",
                                graphitem_RetID(gi));

rd = ask(server, longer_string);

gb_parser(self, rd, typ);

}  /* end instanceof procedure */
/******************* Required housekeeping procedures ************************/

/* initializes remap class */
boolean remap__InitializeObject(ClassID, self)
struct classheader *ClassID;
struct remap *self;
   {
         port = atoi(portnumber); /* ascii-to-integer conversion of global
                                     variable "portnumber" and assignment to
                                     variable "port" */
/* connects server. Note that machine name is the only non-portable argument */
         if(connect_CB_server(port,"isr1","X_11GraphBrowser",
                              strdup(getenv("USER")), &server)!=0)
       {
       }
     return TRUE;
   }

void remap__FinalizeObject(ClassID, self)
struct classheader *ClassID;
struct remap *self;
   {
/*     disconnect_CB_server(server);*/
   }

boolean remap__InitializeClass(ClassID)
struct classheader *ClassID;
   {
     struct classinfo *cbGvtype;

     printf ("Entered initialize...");

     cbGvtype = class_Load("cbGraphView");

     proctable_DefineProc("remap-query", remap_query, cbGvtype,
                          NULL, "Allows retrieval of REMAP objects");
     proctable_DefineProc("remap-instanceof", remap_instanceof, cbGvtype,
                          NULL, NULL);
     printf ("done.\n");
     return TRUE;
   }

void remap_foobarTest (struct cbGraphView *self, long rock)
{
  printf ("You found me!\n");
}
/******************* End "remap.c" source code file *************************/
```

LIST OF REFERENCES

Barkakati, Nabajyoti, *X Window System Programming*, Sams, 1991.

Brown, Laure, "Graphical User Interfaces, A developer's Quandry," *Patricia Seybold's Unix in the Office*, pp. 1-11, August 1989.

Borenstein, Nathaniel S., *Multimedia Applications Development with the Andrew Toolkit*, Prentice Hall, Inc., 1990.

Conklin, Jeff, and Begeman, Michael, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions Office Information Systems*, pp. 303-331, October 1988.

Eherer, Stefan, and Baumeister, Markus, *Documentation of the Andrew Classes cbGraph and cbGraphView and How to Extend the Functionality of the Standard Graphbrowser*, University of Aachen, 1991.

Eherer, Stefan, and Gallersdorfer, Rainer, *ConceptBase V3.1 Programmer's Manual*, University of Aachen, 1992.

Jarke, Matthias, *ConceptBase V3.2 User Manual*, University of Aachen, 1993.

Jones, Oliver, *Introduction to the X Window System*, Prentice Hall, Inc., 1989.

Ramesh, Balasubramanian, and Dhar, Vasant, "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," *IEEE Transactions on Software Engineering*, pp. 498-510, June 1992.

Stenzoski, Jeffrey, *X Window Application Extension with the Andrew Toolkit*, Naval Postgraduate School, September 1992.

Wolf, Andrew D., "The X Windows System, Where is Its Future?," *Patricia Seybold's Unix in the Office*, pp. 3-16, April 1992.

BIBLIOGRAPHY

Carnegie Mellon University, Andrew Toolkit version 5.1 online documentation, 1992.

Jas, Frank, and Russell, Will, *C Language Programming: An Intensive Study*, University of California at Santa Cruz.

Kernigan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, 1978.

University of Aachen, *ConceptBase V3.2* online documentation, 1993.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	B. Ramesh, Code AS/RA Naval Postgraduate School Monterey, California 93943-5002	3
4.	Roger Stemp, Code CS/SP Naval Postgraduate School Monterey, California 93943-5002	1
5.	LT Joseph A. Martinelli 5431 Tejon Street Denver, Colorado 80221	2